CONNECTING SOLUTIONS

Beaglebone cape GPS/GPRS

BBCapeGPSGPRS

User Manual

Code	**:**	BBCapeGPSGPRS-MANUAL-EACH-1.0
Version	**:**	1.0
Date	**:**	20/12/2014

Page intentionally left blank

Warranty, Warnings and Disclaimers

This evaluation board/kit is intended for use for ENGINEERING DEVELOPMENT, DEMONSTRATION, OR EVALUATION PURPOSES ONLY and is not considered by EXPLOITSYS to be a finished end product fit for general consumer use. Persons handling the product(s) must have electronics training and observe good engineering practice standards. As such, the goods being provided are not intended to be complete in terms of required design-, marketing-, and/or manufacturing-related protective considerations, including product safety and environmental measures typically found in end products that incorporate such semiconductor components or circuit boards. This evaluation board/kit does not fall within the scope of the European Union directives regarding electromagnetic compatibility, restricted substances (RoHS), and therefore may not meet the technical requirements of these directives or other related directives.

Please read the User Manual and, specifically, the Warnings and Restrictions notice in the Systems Reference Manual of the BeagleBone you are using, prior to handling the product. This notice contains important safety information about temperatures and voltages.

No license is granted under any patent right or other intellectual property right of Supplier covering or relating to any machine, process, or combination in which such Supplier products or services might be or are used. The Supplier currently deals with a variety of customers for products, and therefore our arrangement with the user is not exclusive. The Supplier assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein.

The BB-BONE-GPS-GPRS as purchased is warranted against defects in materials and workmanship for a period of 90 days from purchase. This warranty does not cover any problems occurring as a result of improper use, modifications, exposure to water, excessive voltages, abuse, or accidents. All boards will be returned via standard mail if an issue is found. If no issue is found or express return is needed, the customer will pay all shipping costs.

Page intentionally left blank

Revision History

Version	Changes	Date	Approved
1.0	Document creation	17/06/2013	DCarona

Page intentionally left blank

Table of Contents

Page intentionally left blank

List of Figures

List of Tables

Page intentionally left blank

1. INTRODUCTION

The BeagleBone Cape described in this document adds GPS and GSM capabilities to the BeagleBone making it suitable for tracking and M2M scenarios. The cape uses Telit GE864, a high quality module with proven performance and reliability. This cape supports the Quad band GSM/GPRS (850 MHz, 900MHz, 1800MHz and 1900MHz).

1.1. Scope

This document describes the cape's behaviors and features, and also how to interact with it.

1.2. Acronyms and Abbreviations

The acronyms and abbreviations used in the present document are:

Acronym	Description
BB	BeagleBone
BBB	BeagleBone Black
GPRS	General Packet Radio Service
GPS	Global Positioning System
I/O	Input/Output
PPPD	Point-to-Point Protocol Daemon
RD	Reference Document
BB-BONE-GPS-GPRS	BeagleBone Cape with GPS and GPRS functionality enable by telit module GE864. (The cape is described in this document)

2. RELATED DOCUMENTS

2.1. Reference Websites

[1] "PPPD - Linux man page," [Online]. Available: http://linux.die.net/man/8/pppd. [Accessed 06 2013].

2.2. Reference documents

Table 2-1 specifies which reference documents should be considered when following this document.

Table 2-1 – Related Documents

Reference	Title	Version	Date
[RD 1]	BeagleBone White System Reference Manual	Rev A5	02/2012
[RD 2]	BeagleBone Black System Reference Manual	Rev A5.2	04/2013
[RD 3]	GE864-GPS Hardware User Guide	Rev. 10	24-04-2012

3. POWER SUPPLY

When BB-BONE-GPS-GPRS cape is connected in your beaglebone the USB connection is not enough to supply both boards so is advisable to use an generic charger with 5V with a minimum current of 1A.

4. BEAGLEBONE BLACK COMPATIBILITY

The BB-BONE-GPS-GPRS is compatible with BeagleBone Black, however, HDMI needs to be disabled due to hardware resource conflicts.

It is possible to disable HDMI on boot through editing “uEnv.txt” file present on BeagleBone Black file system when it is plugged in on computer USB port. To do it simply add the following information into file:

```
optargs=quiet capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN
```

After adding this line save the file and reboot the BB Black.

5. PIN DESCRIPTION

This cape has been developed to be fully compatible with both versions of BeagleBone, therefore based on [RD 1] and [RD 2] the pin map **from BB perspective (I/O)** for both versions is the following:

Table 5-2 – P8 Header pin usage.

Pin	Name on Cape	Name on BBW	Name on BBB	Pin to Export	Mode	Direction
P8.39	STAT_LED	GPIO2_12	LCD_DATA6	76	7	I
P8.40	PWRMON	GPIO2_13	LCD_DATA7	77	7	I
P8.41	GSM_ON_OFF	GPIO2_10	LCD_DATA4	74	7	O
P8.30	GPS_ON_OFF *	GPIO2_25	LCD_DE	89	7	O
P8.31	GSM_CTS	UART5_CTSN	LCD_DATA14	-	6	I
P8.32	GSM_RTS	UART5_RTSN	LCD_DATA15	-	6	O
P8.37	GSM_TXD	UART5_TXD	LCD_DATA8	-	4	O
P8.38	GSM_RXD	UART5_RXD	LCD_DATA9	-	4	I
P8.42	GPS_WAKEUP *	GPIO2_11	LCD_DATA5	75	7	I

Table 5-3 – P9 Header pin usage.

Pin	Name on Cape	Name on BB	Name on BBB	Pin to Export	Mode	Direction
P9.12	PWR_ON_OFF	GPIO1_28	GPIO1_28	60	7	O
P9.24	GPS_TX *	UART1_TXD	UART1_TXD	-	0	O
P9.26	GPS_RX *	UART1_RXD	UART1_RXD	-	0	I

* Only for version with GPS, otherwise consider it free to be used as generic GPIO by BeagleBone.

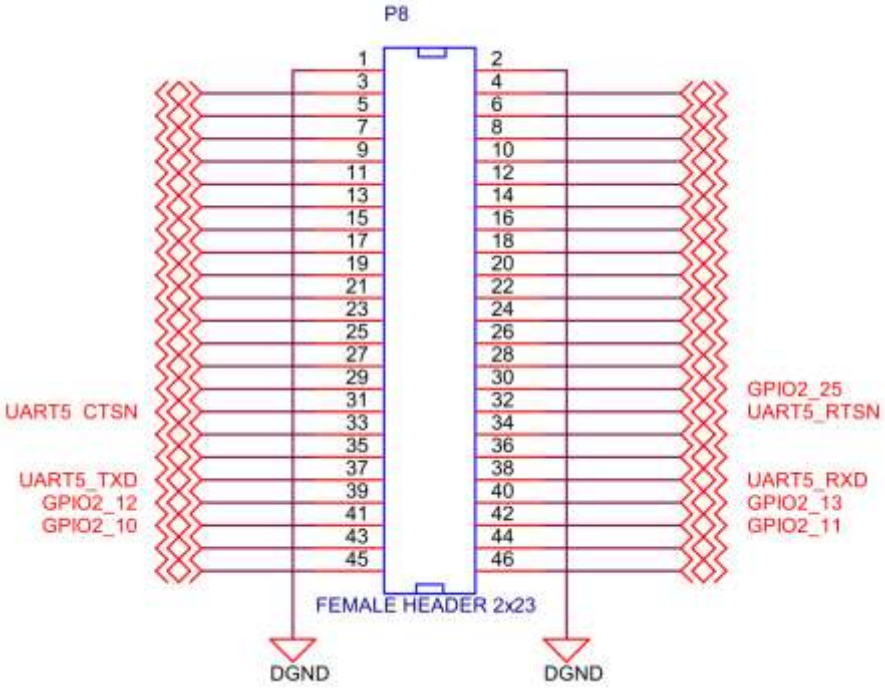

Fig. 5-1 – P8 Pin location on BeagleBone White.

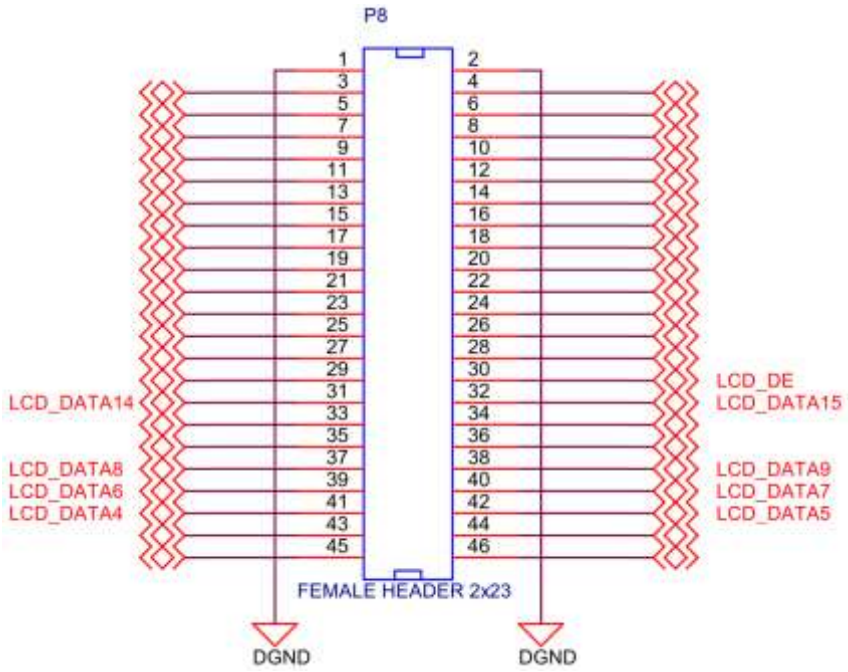

Fig. 5-2 – P8 Pin location on BeagleBone Black.

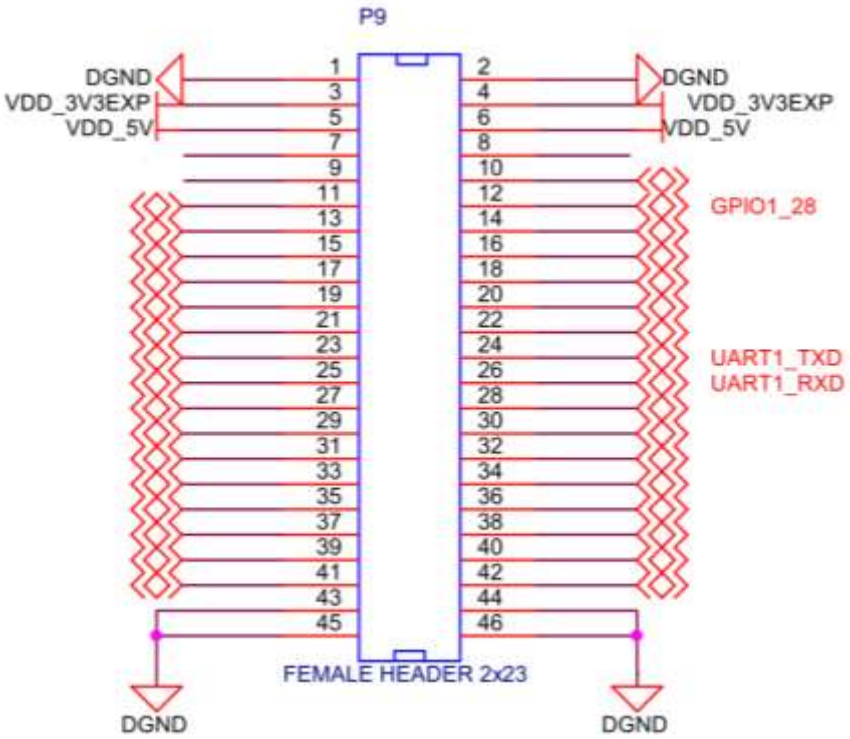

Fig. 5-3 – P9 Pin location in BeagleBone White and Black.

5.1. STAT_LED

This output pin provides information regarding the status of the GSM operating mode. This pin has three different behaviors enumerated in the next table:

Table 5-4 – STAT_LED pin behaviour.

Pin Behavior / Led	Device status
Permanently off	Device off
ON/OFF with period of 1s	Net search / not registered / turning off
ON/OFF with period of 3s	Registered full service

This pin is informational only, and its usage is optional and the led present in the cape is connected to this pin so the user can easily understand the behavior of the cape.

5.2. PWRMON

This output pin can be used to check if the device is turned on. After the module has been switched on, this pin will be raised up after a delay of 1000ms if the power up sequence is successful. This pin goes low when the device is powered off.

5.3. GSM_ON_OFF

This output pin is used to turn on and off the GSM functionality. An active low pulse with duration of at least 1 second toggles the state of the GSM.

5.4. GPS_ON_OFF and GPS_WAKEUP

These two pins are used to control the power of the GPS part of the module **only for version with GPS**.

5.5. PWR_ON_OFF

This pin is **active low** and controls the power supply of the entire cape. When this is low the cape is ON and when asserted the cape is set to OFF.

IMPORTANT NOTE:

Due to conflicts with boot pins, the cape is by default set to OFF, so to start interacting the user must set PWR_ON_OFF pin to zero through the process described in chapter 6 or running the script "init cape" supplied with the board.

6. INTERACTING WITH GPIO'S

To interact with BB GPIO's, it is required to export the pin so the user can have access to it. In order to know the pin number that corresponds to a certain GPIO the following formula can be used:

$$GPIOx\_y = x*32 + y$$

Example:

GPIO1_6 = 1*32 + 6 = 38

The pin that has to be exported is "pin 38".

After discovering pin number the following procedure should be adopted:

1. Select the folder that controls GPIO's
    a. cd /sys/class/gpio/
2. Choose which pin to export
    a. echo 38 > export
3. Select the exported pin file
    a. cd /sys/class/gpio/gpio38
4. Set pin direction
    a. echo in > direction
5. Read pin
    a. cat value

If pin is set to work as output the procedure is the same but in direction file should be echoed "out" instead of "in" and to affect the pin "1" or "0" should be echoed to value file.

7. INITIAL SETUP

Before plug in BB-BONE-GPS-GPRS cape into user beaglebone for the first time, some steps need to be followed:

7.1. Cape Firmware

- Insert the file *cape-bone-gps-gprs-00A0.dtbo* supplied with the cape into the firmware folder:

```
cp cape-bone-gps-gprs-00A0.dtbo /lib/firmware/
```

7.2. Antennas

The cape has two connectors for antennas where J1 corresponds to GSM antenna and J2 to GPS active antenna.

7.3. GPRS Communication protocol

7.3.1. Setup

In order to start communicating through a GPRS connection it is mandatory to make sure that the PPPD application is installed on the BeagleBone. If you are using Angstrom Linux, the PPP package can be installed with the following command:

```
opkg install ppp
```

BeagleBone requires internet access (Ethernet cable or other) to download and install this application through the command described above. If it is already installed a message reporting that your version is up to date is shown on the user console.

After the correct installation of PPPD a new directory called *ppp* is created on */etc/* directory.

Inside this directory you will need to create a directory named peers:

```
mkdir /etc/ppp/peers
```

Inside this directory you should place the *ppp-start* file supplied with the cape.

After that, copy the files *gprs-connect-chat* and *init_modem* to */etc/ppp* directory.

To conclude configuration process, */etc/ppp/options* file has to be edited and some options changed according with the following:

```
/dev/ttyO5
57600
lock
crtscts
noauth
defaultroute
user <your user name>
password <your password>
connect '/usr/sbin/chat -V -e -t 90 -f /etc/ppp/gprs-connect-chat'
noipdefault
usepeerdns
noccp
debug
logfile /var/log/ppp.log
```

user – your ISP user name.

password – your ISP login password.

Please note that *debug* and *logfile* options are not mandatory, you may use it for debug purposes by tailing *ppp.log* file. For example:

```
tail -f /var/log/ppp.log
Press CTRL+C to exit
```

This file probably won't exist on the specified directory. If you want to use the debug feature you need to create it.

The options suggested are the minimum for module's correct functioning, however users may add more options according with the application scenario. Please refer to [1] for more information on PPPD options.

IMPORTANT NOTE:

It is recommend that the SIM card used in your cape has its PIN code disabled. Some users reported problems when using PPPD with the SIM card PIN code active, so **for correct functioning your SIM card should have its PIN code disabled.**

To start interacting with BB-BONE-GPS-GPRS please refer to chapter 8.

8. INTERACTING WITH CAPE

IMPORTANT NOTE:

Please be aware that beaglebone might take some seconds (about 60s) to start when BB-BONE-GPS-GPRS is plugged in due to EEPROM read, so do not be worried when that happens.

8.1. Manual Process (every time BB boots up)

- **Enable cape:**

```
echo BB-BONE-GPS-GPRS > /sys/devices/bone_capemgr.*/slots
```

After, the cape is enabled on user BeagleBone and ready to be used.

To validate that the operations has succeeded you can type:

```
cat /sys/devices/bone_capemgr.*/slots
```

If the cape firmware is correctly loaded the cape name is shown.

- **Initialize GPRS Modem:**

Due to cape design when it is powered GPRS part is automatically turned on, so the led present on cape should start blinking as soon the cape starts.

- **Starting PPPD**

After setup is completed you are able to start running PPPD using the following command:

```
. /etc/ppp/peers/ppp-start
```

This command will start the process that will connect your BB to this module and consequently to the Internet.

- **Testing PPPD connection**

If the process finished successfully, when you execute the *ifconfig* command you should see a new interface called *ppp0* on your networks interface list.

To test if everything is correctly running you can attempt to ping an IP address that is expected to respond.

- **Stopping PPPD (optional)**

To stop PPPD process you can simply run the following commands:

1. To remove *ppp0* interface:

```
killall pppd
```

To remove the lock process that has been made to ttyO5

```
rm /var/lock/LCK..ttyO5
```

After these two commands you can check that interface *ppp0* has been removed from your network connections list.

- **Stopping GPRS Modem:**

To stop GPRS part of cape the user only needs to run script *init_modem*:

```
/etc/ppp/init_modem
```

This script will stop GPRS part, and to ensure that it have been stopped, led present on cape should stop blinking.

8.1.1. GPS Communication (Only for version with GPS)

Communication with GPS module simply needs to turn on the module and NMEA message starts appearing in ttyO1.

8.1.2. GPS Power-up

To switch ON the GPS a pulse is required at the input pin GPS_ON_OFF. In order to know when the GPS is ready to accept the pulse, the application in the host controller can either:

- Wait for a fixed interval;
- Monitor a pulse on GPS_WAKEUP output;
- Assert a pulse on the GPS_ON_OFF input every second until GPS starts indicating a high on GPS_WAKEUP output or generating serial messages on ttyO1.

The GPS starts after asserting a rising edge pulse on GPS_ON_OFF input and when high level persists for at least three cycles of RTC clock.

To start receiving GPS messages the user only needs to turn on the GPS part of the module by running the script "init_GPS" supplied.

Finally configure ttyO1 UART to work as:

- Data rate: 4800bps;
- Data bits: 8;
- Parity: None;
- Stop bits: 1;
- Flow control: None;

After this you should be able to see GPS messages on ttyO1.

8.1.3. GPS Shutdown

When GPS is working, a pulse on the GPS_ON_OFF input triggers the GPS shutdown sequence. This process may take from 10ms up to 900ms, depending on operation in progress.

To shutdown the GPS part of the module the user can also run "*init_GPS*" script.

8.2. Automatic Process (on Boot)

To setup user BeagleBone to start automatically GPRS and GPS parts the user has to follow the procedure described above:

1. Follow Chapter 6 and confirm that everything works;
2. Insert script "*initcape*" into /etc/init.d folder;
3. In console execute the following command to create a new service to be run at startup:

```
update-rc.d initcape defaults
```

4. Reboot BeagleBone and check if cape has been enabled and GPRS and GPS have been correctly started.

If user needs to remove the service created to cape can simply run the following command:

```
update-rc.d -f initcape remove
```

9. USE CASE EXAMPLE

9.1. Use case description

The use case presented shows how to configure BeagleBone to use BB-BONE-GPS-GPRS cape in a continuous mode to send GPS position through GPRS connection to a server link every 5 min (configurable).

9.2. Required configuration (setup one time use always)

Set your script to run at boot. Please make sure that:

- SIM card is connected to cape with PIN code disabled;
- GPS and GPRS antennas are connected;

9.3. Simple Application code

```
/*****************************
 GPS TEST CAPE SCRIPT
 Created on: Aug 06, 2013
 Author: dcarona
*****************************/

#include "stdio.h"
#include "stdlib.h"
#include "fcntl.h"
#include "termios.h"
#include "string.h"
#include "unistd.h"

#define TRUE 1
#define FALSE 0

int fd;
char buffer[8];

struct Location
{
   char Time[8];
   char status[2];
   char Lat[10];
   char N_S_indicator[2];
   char Long[11];
   char E_W_indicator[2];
   char Speed[9];
   char angle[7];
   char Date[7];
};
```

```
int readLine(char * result);
void parseMSG(char *msg, struct Location *location, int size);
void sendInfoToServer (char *message);
void buildLink (struct Location *location, char * addr);

int main()
{
   int size;
   struct termios opts;
   struct Location location;
   char line[128];
   char address[256];
   int test_send = FALSE;


   /* Attempt to open the serial port */
   if ((fd = open("/dev/ttyO1",O_RDWR | O_NOCTTY | O_NDELAY)) == -1) {
      printf("Unable to open serial device, error %d\n",fd);
   }

   /* Configure serial port */
   tcgetattr(fd,&opts);

   opts.c_cflag = (opts.c_cflag & ~CSIZE) | CS8;     // 8-bit chars
   opts.c_iflag &= ~IGNBRK;         // ignore break signal
   opts.c_lflag = 0;                // no signaling chars, no echo,
                                    // no canonical processing
   opts.c_oflag = 0;                // no remapping, no delays
   opts.c_cc[VMIN]  = 0;            // read doesn't block
   opts.c_cc[VTIME] = 5;            // 0.5 seconds read timeout

   opts.c_iflag &= ~(IXON | IXOFF | IXANY); // shut off xon/xoff ctrl

   opts.c_cflag |= (CLOCAL | CREAD);// ignore modem controls,
                       // enable reading
   opts.c_cflag &= ~(PARENB | PARODD);      // shut off parity
   opts.c_cflag |= 0;
   opts.c_cflag &= ~CSTOPB;
   opts.c_cflag &= ~CRTSCTS;
   cfsetispeed(&opts,B4800);

   tcsetattr(fd,TCSANOW,&opts);
```

```
for (;;) {

      //Delay
      if (test_send == TRUE) {
         sleep(1800); // Delay of 30min = 1800s
         test_send = FALSE;
      }
      size = readLine(line);

      if (strstr(line,"$GPRMC") != 0) {

         test_send = TRUE;
         parseMSG(line, &location, size);

         buildLink(&location, address);
         printf("Address:[%s]\n", address);
         sendInfoToServer(address);

         tcflush(fd, TCIFLUSH);
      }
   }
}
/*
 * Reads one complete line from GPS serial port
 */
int readLine(char * result)
{
   int lineFeedFound =FALSE;
   int crfound = FALSE;
   int i = 0;
   char c;

   while ((lineFeedFound==FALSE) && (i < 128)) {

      if (read(fd,&c,1) > 0) {

         result[i++] = c;

         if (c == 10) {
            result[i++]=0;
            lineFeedFound = TRUE;
         }
      }
   }
   return i;
}
```

```
/*
 * Separates each message into several fields and afects location struct.
 */
void parseMSG(char *msg, struct Location *location, int size)
{

   //$GPRMC,090446.000,A,3845.5509,N,00906.9152,W,0.00,129.54,080813,,,A*79

   char comma[2] = ",";
   char point[2] = ".";
   char *copy_time, *copy_vel;
   char *token;
   char *token_time, *token_vel;
   float vel;

   //printf("Msg:[%s]\n", msg);

   token = strtok(msg, comma);

   token = strtok(NULL, comma);

   copy_time = token;

   token = strtok(NULL, comma);

   if (*token == 'A') {

      //*********** STATUS ************************//
      memcpy(location->status, token, strlen(token));

      //INSERT 0 IN THE END OF STRING SO SYSTEM CAN DETECT ITS END
      location->status[strlen(token)] = 0;

      //*********** LATITUDE **********************//
      token = strtok(NULL, comma);
      memcpy(location->Lat, token, strlen(token));
      location->Lat[strlen(token)] = 0;

      //*********** N S INDICATOR *****************//
      token = strtok(NULL, comma);
      memcpy(location->N_S_indicator, token, strlen(token));
      location->N_S_indicator[strlen(token)] = 0;

      //*********** LONGITUDE *********************//
      token = strtok(NULL, comma);
      memcpy(location->Long, token, strlen(token));
      location->Long[strlen(token)] = 0;

      //*********** E W INDICATOR *****************//
      token = strtok(NULL, comma);
      memcpy(location->E_W_indicator, token, strlen(token));
      location->E_W_indicator[strlen(token)] = 0;

      token = strtok(NULL, comma);
```

```
      copy_vel = token;
      //*********** ANGLE *************************//
      token = strtok(NULL, comma);
      memcpy(location->angle, token, strlen(token));
      location->angle[strlen(token)] = 0;

      //*********** DATE **************************//
      token = strtok(NULL, comma);
      memcpy(location->Date, token, strlen(token));
      location->Date[strlen(token)] = 0;

      //*********** TIME **************************//
      token_time = strtok(copy_time, point);
      memcpy(location->Time, token_time, strlen(token_time));
      location->Time[strlen(token_time)] = 0;

      //*********** VELOCITY **********************//
      token_vel = strtok(copy_vel, point);
      vel = atof(token_vel);
      vel = vel * 1.852;
      sprintf(location->Speed, "%f", vel);
      location->Speed[strlen(token_vel)] = 0;
   }
}
/*
 * Sends position information to a specified server
 */
void sendInfoToServer (char *message) {
   system(message);
}
/*
 * Builds link to use in the call of server info update according with the
specified format
*/
void buildLink (struct Location *location, char * address) {

   strcpy(address, "curl “your link”,");
   strcat(address, location->Lat);
   strcat(address, ",");
   strcat(address, location->N_S_indicator);
   strcat(address, ",");
   strcat(address, location->Long);
   strcat(address, ",");
   strcat(address, location->E_W_indicator);
   strcat(address, ",");
   strcat(address, location->Date);
   strcat(address, ",");
   strcat(address, location->Time);
   strcat(address, ",");
   strcat(address, location->Speed);
   strcat(address, ",");
   strcat(address, "0");
}
```